Software Requirements Specification

For

Uber Based Ride booking Application

Version 1.0

Prepared By:

Tanmay Mehrotra

Nitish Kumar

Shailja Chaurasia

Anshu Kumar

Index

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to provide a detailed description of the functionality, design, and performance criteria for the Cab Ride Booking and Payment Application. This app allows users to book rides, make payments, and track their journey in real-time

1.2 Scope

The Cab Ride Booking and Payment Application will provide the following key features:

- **User Registration & Authentication**: Allows users to create an account and securely log in.
- **Ride Booking**: Allows users to book a cab based on their current location or a manually entered address
- **Ride Tracking**: Allows users to track their booked ride in real-time.
- **Payment**: Enables users to make payments for their rides via different methods (credit card, digital wallets, etc.).
- **Ride History**: Provides users with the history of past rides and payment details.

1.3 Definitions, Acronyms, and Abbreviations

- **Cab**: A vehicle available for hire by users for transportation.
- **User**: A person who uses the app to book rides and make payments.
- **Payment Gateway**: A service that processes online payments.
- **Admin**: The person responsible for managing the app and overseeing user complaints, service issues, etc.
- **ETA**: Estimated Time of Arrival.

2. Overall Description

The Cab Ride Booking and Payment Application is a react based application intended to provide on-demand cab booking services to users. The application will integrate with third-party services for map routing, real-time tracking, and payment processing.

2.1 Product Features

- **User Registration & Login**: Users can sign up and authenticate using email or mobile login.
- **Cab Ride Booking**: Allows users to input their pickup and drop-off locations and book a ride with multiple options of type of vehicle to use along with the respective ride cost.
- **Ride Tracking**: Provides users with real-time updates on the location and ETA of their cab.
- **Payment System**: Multiple payment methods (credit/debit cards, wallets, cash on delivery).
- **Ride History**: Access to previous ride details, payments, and ratings.
- **Notifications**: Push and email notifications for confirmations, status updates, etc

3. Technical Specifications

3.1 Functional Requirements

3.1.1 User Registration and Authentication

- The system must allow users to sign up using an email address or mobile number.
- Users must be able to log in using their registered email address or mobile number.
- Passwords must be securely stored using encryption.
- Users must be able to reset their passwords via email or mobile number.

3.1.2 Ride Booking

- The user must be able to enter a pickup and drop-off location.
- The system must estimate the fare based on the distance and time of day.
- The system must provide the user with available cab options (e.g.,sedan, SUV).
- The user must be able to select the desired cab type and confirm the booking.
- The system must notify the user if there is a change in the status of their booking (e.g., cancellation, delay).

3.1.3 Payment System

- The system must support payment using PayPal.
- The system must provide a secure interface for adding, updating, and deleting payment methods.
- The user should be able to make payments after the completion of their ride and receive a receipt via email or push notification.
- Users should receive real-time notifications about payment success, failure, and transaction details.

3.1.4 Ride History

- Users should be able to view a history of their past rides, including ride details like date, time, distance, fare, and payment method.
- Users should be able to view a history of their past rides, including ride details like date, time, distance, fare, and payment method.
- Users should be able to download or view receipts for their past transactions.

3.2 Non-Functional Requirements

3.2.1 Performance

- The application should handle up to 10,000 concurrent users during peak hours without significant latency.
- All responses (ride bookings, payments, notifications, etc.) should be processed within 3 seconds under normal conditions.
- The application should provide a smooth, lag-free experience for users during ride tracking.

3.2.2 Scalability

- The application should be scalable to handle future growth, especially during peak demand periods (e.g., holidays, rush hours).
- The backend architecture should be modular and support horizontal scaling to accommodate increasing user base and transactions.

3.2.3 Security

- The system must ensure secure transmission of data, using encryption for sensitive data both in transit and at rest.

- All sensitive data (such as passwords and payment details) must be encrypted using modern cryptographic algorithms (e.g., TLS/SSL).

- User passwords must be securely hashed and salted before being stored.

3.2.4 Availability and Reliability

- The system should ensure 99.9% uptime with backup mechanisms in place in case of system failure.
- The system should be able to recover from a failure within 5 minutes.

3.2.5 Usability

- The system should provide an intuitive, easy-to-navigate user interface for users with varying levels of technical expertise.
- The system should include a help section or user manual to guide new users in interacting with the platform.

3.3 Feasibility Study

3.3.1. Technical Feasibility

➢ **Benefits**

❖ **Frontend (React):**

- **Component-Based Architecture:** React's modular and reusable component system enables quick development and easier maintenance of UI components, enhancing the speed of development for features like booking forms, user profiles, and maps.
- **High Performance**: React’s Virtual DOM provides efficient updates and rendering, making it ideal for applications that require real-time user interaction, such as ride status updates and live ride tracking.
- **Cross-Platform Compatibility:** With the help of React Native, a mobile version of the ride-booking app can be easily developed for both Android and iOS, sharing a significant portion of the codebase, reducing development time and effort.

- **Active Ecosystem**: React has a large and active developer community, providing a wide range of libraries, tools, and plugins (like React Router, Redux, and Material UI) to enhance functionality and speed up development.

❖ **Backend (Flask):**

- **Lightweight and Flexible:** Flask is a lightweight web framework that provides the flexibility to structure the backend as needed. It is easy to scale and customize for the needs of a ride-booking application.
- **RESTful API Support:** Flask is well-suited for building RESTful APIs, which is a core requirement for a ride-booking application. APIs can be used to handle requests for ride booking, user authentication, payment processing, and location services.
- **Integration with Databases**: Flask integrates well with databases like PostgreSQL, MySQL, or MongoDB, allowing efficient storage and retrieval of user, ride, and payment data.

❖ **Scalability:**

- **Horizontal Scaling:** Flask's lightweight nature makes it easier to scale horizontally by running multiple instances behind a load balancer, which is beneficial for handling increased traffic.
- **React and Flask Decoupling:** Separating the frontend (React) and backend (Flask) allows independent scaling of both. For instance, if the frontend faces increased traffic (e.g., during peak booking hours), the backend can still function optimally with appropriate resource allocation.

❖ **Third-Party Service Integrations:**

- Payment Gateways: React and Flask can integrate seamlessly with popular payment systems like Stripe, PayPal, or Razorpay, enabling secure in-app payments for rides.

➢ **Challenges:**

❖ **Real-Time Features:**

- **Real-Time Communication:** Although Flask supports real-time communication via Flask-SocketIO, maintaining real-time updates for ride tracking, driver availability, and booking status can introduce complexities, especially when scaling the app to a large user base. Managing WebSockets efficiently across multiple connections could lead to performance issues.

- **Handling Traffic Spikes:** Flask’s single-threaded nature can sometimes cause bottlenecks during high traffic periods. To scale effectively, additional solutions like caching (Redis) or task queues (Celery) may be required.

❖ **Performance and Latency:**

- **Frontend Performance:** While React provides high performance, handling large amounts of dynamic data (such as ride status updates or real-time maps) may require optimization techniques like lazy loading, pagination, and debouncing to prevent UI lag.

- **Backend Performance**: Flask can face challenges in handling a large number of concurrent requests, particularly during peak periods, due to its synchronous nature. Solutions like using Nginx or Gunicorn with Flask for load balancing and concurrency would be needed to handle more users and requests.

❖ **Database Management:**

- **Data Consistency:** Flask may require additional configurations to ensure data consistency and handle complex relationships between users, rides, and payments. Managing the transactions across these entities, especially when scaling, could lead to issues if not handled correctly.

- **Database Integration:** While Flask integrates well with databases, ensuring fast and efficient queries as the system grows (especially in terms of real-time data like ride locations and statuses) may require optimization techniques like indexing or query caching.

3.3.2. Non-Technical Feasibility

➢ **Benefits**

❖ **User Adoption and Experience:**

- React’s component-based structure ensures user-friendly, interactive interfaces.
- React Native allows a single codebase for both Android and iOS, ensuring cross-platform availability.

❖ **Cost-Effectiveness:**

- React and Flask’s modular approach reduces development time and costs.
- Open-source nature lowers licensing and software costs, with community-driven support.

❖ **Maintenance and Updates:**

- Decoupled architecture allows independent frontend and backend updates.
- Active communities for React and Flask provide resources for ongoing maintenance.

❖ **Market and Business Potential:**

- Real-time features like ride tracking and driver availability enhance user satisfaction.
- Integration with third-party services (payments, maps) improves the app's competitiveness.

➢ **Challenges:**

❖ **User Engagement:**

- Real-time communication management can be resource-intensive, affecting performance.
- Continuous value offering is crucial for user retention, such as loyalty features and bug-free experiences.

❖ **Business Scalability:**

- Handling traffic spikes during peak hours requires robust infrastructure and investment.
- International expansion may introduce challenges with localization and payment systems.

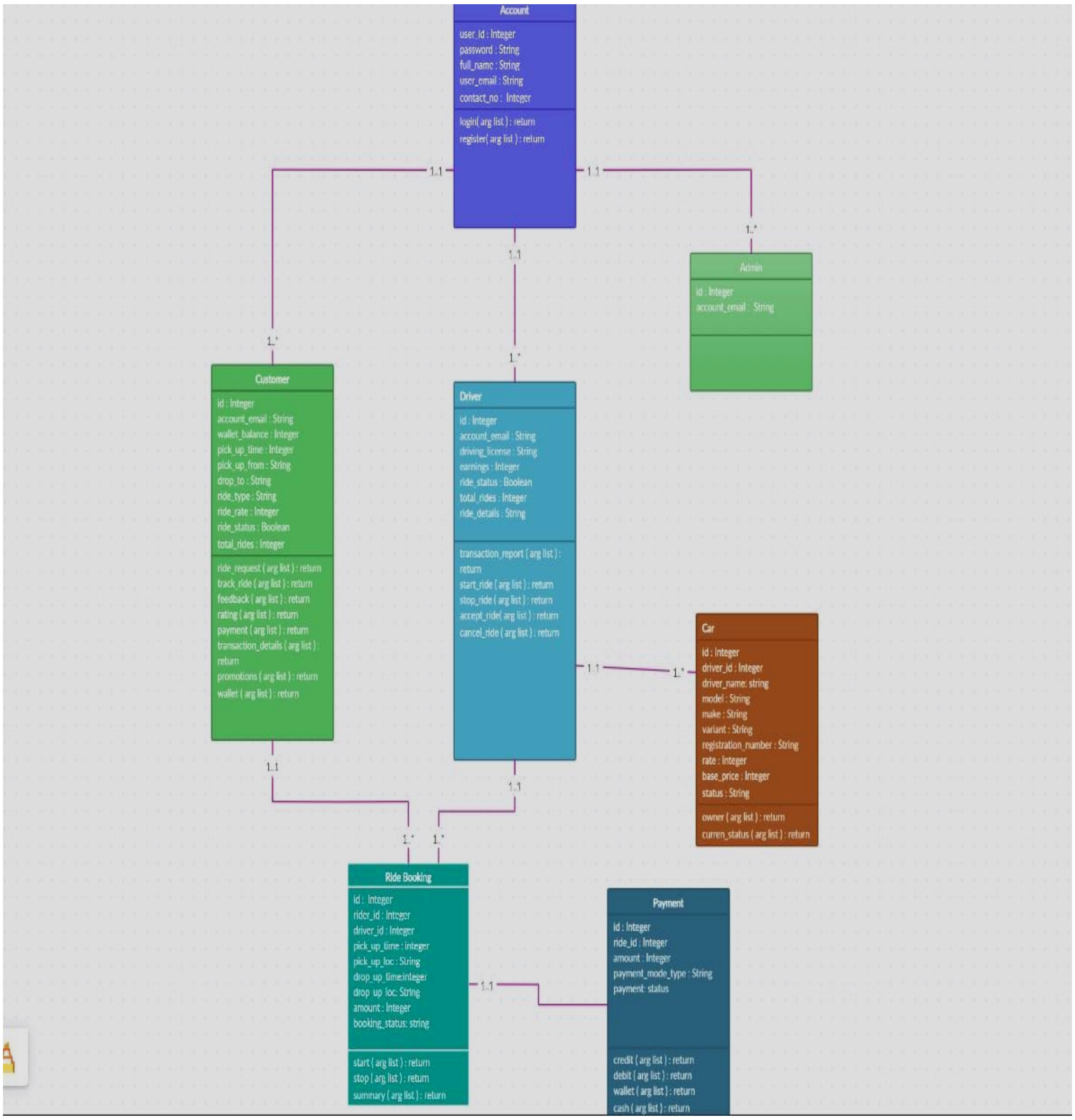
❖ **Security and Privacy Concerns:**

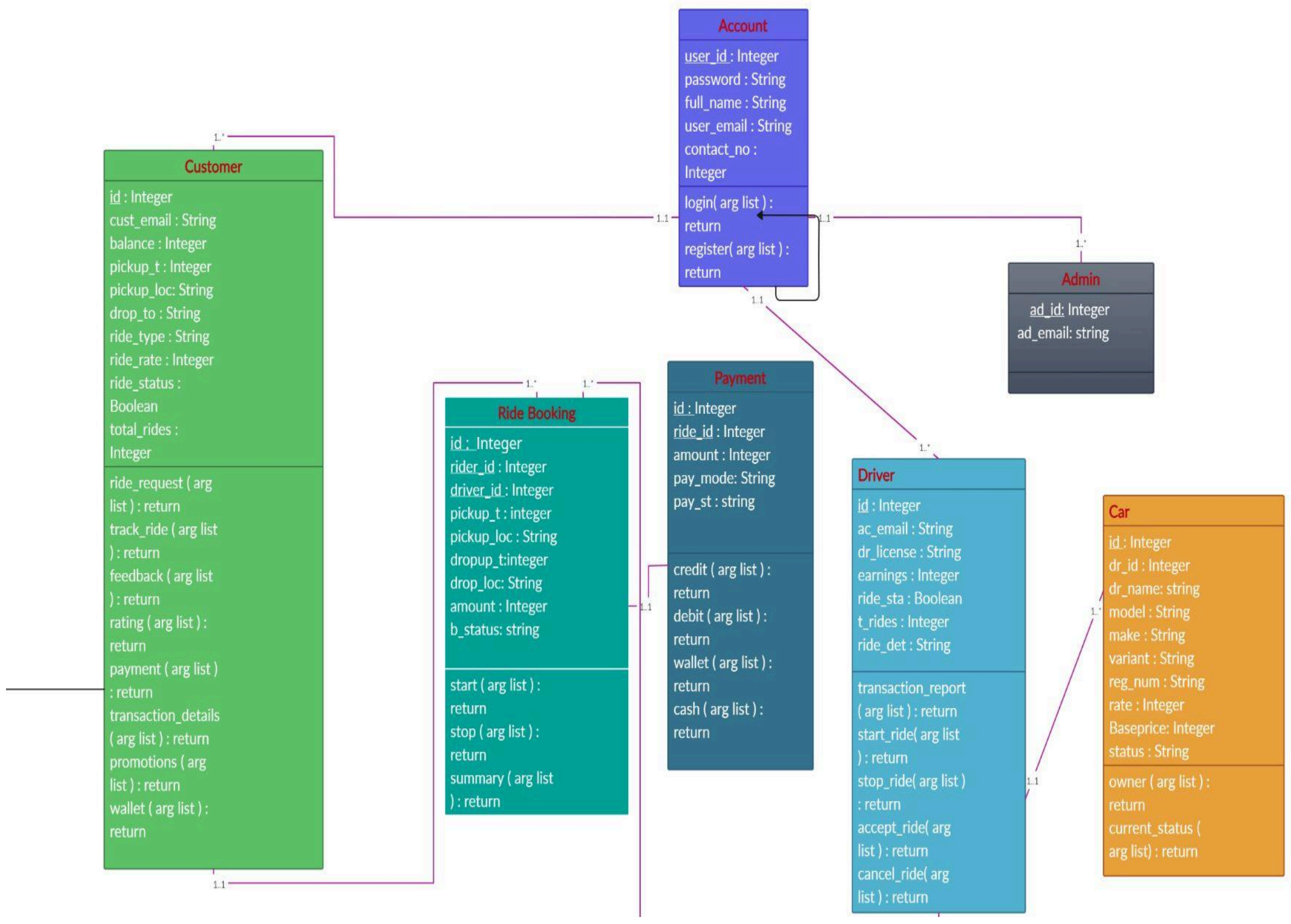
- Sensitive data handling requires strict security measures like SSL encryption and compliance with data protection laws.
- Third-party integrations must be vetted for security risks.

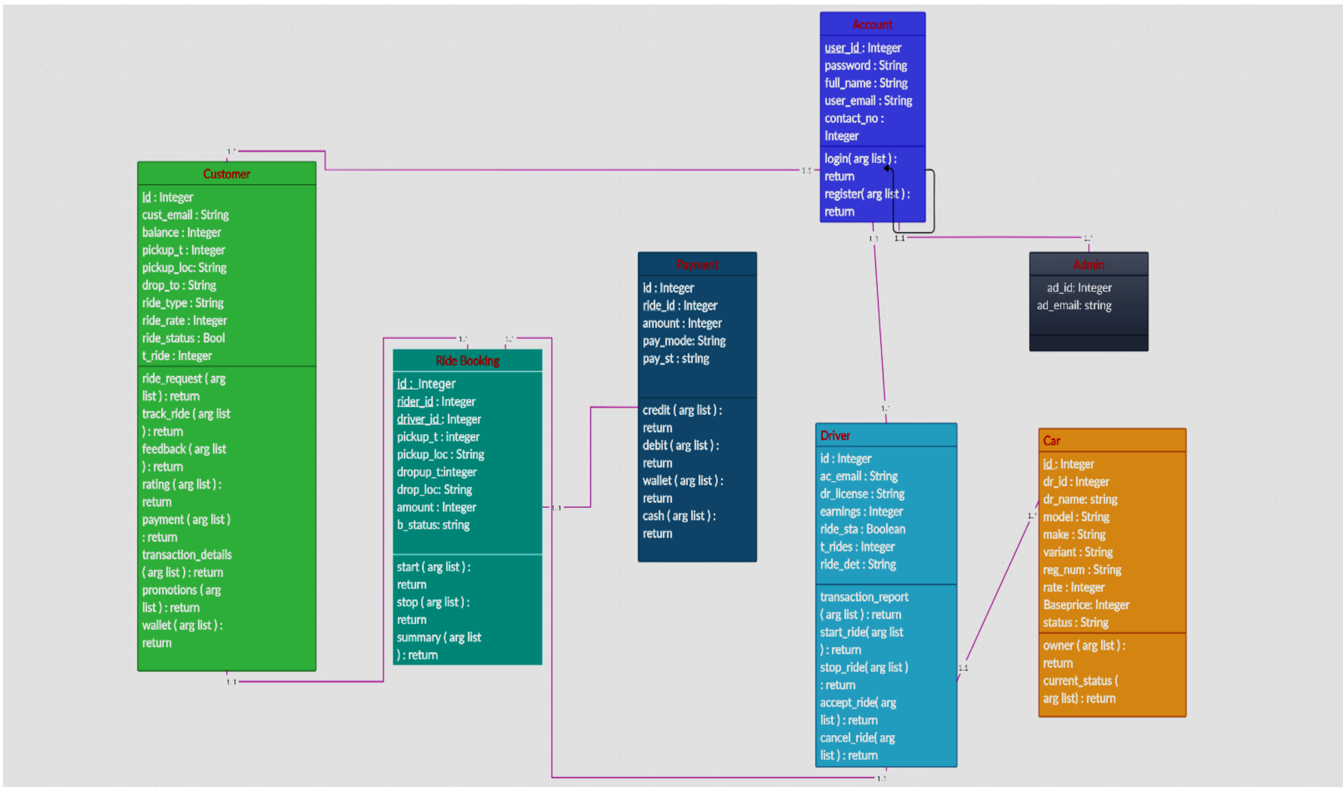
3.4 Software Requirements

- Visual Studio Code
- Pycharm

4. Data Requirements


Account
user_id : Integer
password : String
full_name : String
user_email : String
contact_no : Integer
login(arg list) : return
register(arg list) : return
1..1
1..1
1..1
1..*
1..*
1..*
Admin
id : Integer
account_email : String
Customer
id : Integer
account_email : String
wallet_balance : Integer
pick_up_time : Integer
pick_up_from : String
drop_to : String
ride_type : String
ride_rate : Integer
ride_status : Boolean
total_rides : Integer
ride_request (arg list) : return
track_ride (arg list) : return
feedback (arg list) : return
rating (arg list) : return
payment (arg list) : return
transaction_details (arg list) : return
promotions (arg list) : return
wallet (arg list) : return
Driver
id : Integer
account_email : String
driving_license : String
earnings : Integer
ride_status : Boolean
total_rides : Integer
ride_details : String
transaction_report (arg list) : return
start_ride (arg list) : return
stop_ride (arg list) : return
accept_ride(arg list) : return
cancel_ride (arg list) : return
1..1
1..*
Car
id : Integer
driver_id : Integer
driver_name: string
model : String
make : String
variant : String
registration_number : String
rate : Integer
base_price : Integer
status : String
owner (arg list) : return
curren_status (arg list) : return
1..1
1..1
1..*
1..*
Ride Booking
id : Integer
rider_id : Integer
driver_id : Integer
pick_up_time : integer
pick_up_loc : String
drop_up_time:integer
drop_up_loc: String
amount : Integer
booking_status: string
start (arg list) : return
stop (arg list) : return
summary (arg list) : return
1..1
Payment
id : Integer
ride_id : Integer
amount : Integer
payment_mode_type : String
payment: status
credit (arg list) : return
debit (arg list) : return
wallet (arg list) : return
cash (arg list) : return


Account
user_id : Integer
password : String
full_name : String
user_email : String
contact_no : Integer
login(arg list) : return
register(arg list) : return
Customer
id : Integer
cust_email : String
balance : Integer
pickup_t : Integer
pickup_loc: String
drop_to : String
ride_type : String
ride_rate : Integer
ride_status : Boolean
total_rides : Integer
ride_request (arg list) : return
track_ride (arg list) : return
feedback (arg list) : return
rating (arg list) : return
payment (arg list) : return
transaction_details (arg list) : return
promotions (arg list) : return
wallet (arg list) : return
Admin
ad_id: Integer
ad_email: string
Ride Booking
id : Integer
rider_id : Integer
driver_id : Integer
pickup_t : integer
pickup_loc : String
dropup_t:integer
drop_loc: String
amount : Integer
b_status: string
start (arg list) : return
stop (arg list) : return
summary (arg list) : return
Payment
id : Integer
ride_id : Integer
amount : Integer
pay_mode: String
pay_st : string
credit (arg list) : return
debit (arg list) : return
wallet (arg list) : return
cash (arg list) : return
Driver
id : Integer
ac_email : String
dr_license : String
earnings : Integer
ride_sta : Boolean
t_rides : Integer
ride_det : String
transaction_report (arg list) : return
start_ride(arg list) : return
stop_ride(arg list) : return
accept_ride(arg list) : return
cancel_ride(arg list) : return
Car
id : Integer
dr_id : Integer
dr_name: string
model : String
make : String
variant : String
reg_num : String
rate : Integer
Baseprice: Integer
status : String
owner (arg list) : return
current_status (arg list) : return
1.*
1.1
1.1
1.*
1.1
1.*
1.*
1.*
1.1
1.1
1.*
1.1



5. External Interface Requirements

5.1. API Interfaces

- The system should provide an API tools PostmanAPI,Swagger,ThunderBuilt for third-party applications.
- The API should support both REST to accommodate different use cases.